Meta-Lamarckian-based Iterated Greedy for Optimizing Distributed Two-stage Assembly Flowshops with Mixed-Setup

Pourya Pourhejazy[1], Chen-Yang Cheng[2], Kuo-Ching Ying[*,2], Nguyen Hoai Nam[2,3]

[1]*Department of Industrial Engineering, UiT- The Arctic University of Norway, Lodve Langsgate 2, Narvik 8514, Norway*

[2]*Department of Industrial Engineering and Management, National Taipei University of Technology, Taipei 106, Taiwan*

[3]*Gudeng Precision Industrial Co., LTD, New Taipei City 236, Taiwan*

ABSTRACT

Integrated scheduling of distributed manufacturing operations has implications for supply chain optimization and requires further investigations to facilitate its application area for various industry settings. This study extends the limited literature of the distributed two-stage production-assembly scheduling problems offering a twofold contribution. First, an original mathematical extension, the Distributed Two-Stage Assembly Flowshop Scheduling Problem with Mixed-Setup (DTSAFSP-MS) is investigated to integrate setup time constraints while addressing an overlooked scheduling assumption. Second, a novel extension to the Iterated Greedy algorithm is developed to solve this understudied scheduling problem. An extensive set of test instances is considered to evaluate the effectiveness of the developed solution algorithm comparing it with the current-best-performing algorithm in the literature. Results are supportive of the Meta-Lamarckian-based Iterated Greedy (MIG) as a strong benchmark algorithm for solving DTSAFSP-MS with the statistical tests confirming its meaningfully better performance compared to the state-of-the-art.

Keywords: Production management; Distributed manufacturing; Two-stage assembly flowshop; Makespan; Metaheuristics

[*] Corresponding Author. Tel.: +886 2 2771 2171; fax: +886 2 2731 7168

E-mail address: kcying@ntut.edu.tw (K.-C. Ying).

1. Introduction

The competitiveness of companies depends on the performance of their supply chains (SCs). Given the interactions between various entities across SCs, the operational activities should not be planned in a stand-alone way; the players should follow a system-wide strategy to pursue the global best of the operational performance (Simchi-Levi et al. 2008). Production scheduling helps coordinate the production operations across SCs (Ying et al. 2020); integrated planning approaches can bring about improvements beyond optimality norms for individuals, which results in collective productivity.

Production scheduling literature has witnessed substantial development with myriad mathematical extensions and solution algorithms to address the basic assumptions of the scheduling theory that limits its practicability and real-world applications (Fazel Zarandi et al. 2020; Parente et al. 2020). Introducing job- and process-related constraints and new decision variables have been at the center of attention for bridging the gap between scheduling theory and practice. Integrated production-assembly scheduling is a prime example of growing popularity, which has evolved from the Two-Stage Assembly Scheduling Problem (TSASP; Lee et al., 1993) and the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP; Hatami et al., 2013), to integrate heterogeneous operations across distributed manufacturing facilities.

In the industries where the Original Equipment Manufacturers (OEM) produce components and parts for a final assembly at the assembly centers, DAPFSP is a suitable optimization tool. In many other cases, however, the manufactured components should be assembled soon after the production process because of the operational limitations and product features, for example, when sealant/adhesives must be applied before the final assembly. In this situation, the Distributed Two-Stage Assembly Flowshop Scheduling

Problem (DTSAFSP; Xiong & Xing, 2014) best describes the real operational environment. The DTSAFSPs studies are comparatively limited; only a handful of published works have contributed to the development of this scheduling extension in flowshops with more than three machines (Deng et al. 2016; Lei et al. 2020; Xiong et al. 2014; Xiong and Xing 2014; G. Zhang and Xing 2018).

Accounting for setup operations has been a major improvement in the scheduling field and a prime example of efforts for bridging the gap between theory and practice. Recent studies integrated the setup constraints into various classes of distributed scheduling problems (Hatami et al. 2015; J.-P. Huang et al. 2020, 2021; Y. Li, Li, Gao, and Meng 2020; Y. Li, Li, Gao, Zhang, et al. 2020; Meng and Pan 2021; Xiong et al. 2014; Ying et al. 2020); these studies assumed that all of the operations require setup operations, which may not be the case when integrated production-assembly are considered. To put it into context, let assume the situation where a part of the production process is done using dedicated machinery, robots, or human operators, like assembling parts, while the rest of the machines are multi-purpose. In this situation, the parts arriving for final assembly can be processed without requiring a setup. In this example, considering a mixed-setup setting account for operations with and without setup time constraints to better represent the situation. This special case of having Sequence-Dependent Setup Times (SDST) is prevalent in the scheduling environments with heterogeneous operations, e.g., parts production, assembly, and service operations. Motivated by this practical setting, this study proposes an original mathematical formulation for the Distributed Two-Stage Assembly-Flowshop Scheduling Problem with Mixed Setups (DTSAFSP-MS) and develops a novel metaheuristic to effectively solve the problem.

The remainder of this manuscript is structured in four sections. Section 2 provides an exhaustive review of the relevant literature. Section 3 elaborates on the proposed method, i.e., the mathematical formulation and the solution algorithm. Section 4 is dedicated to

experimental results analysis to benchmark the developed solution method against the state-of-the-art algorithm developed to solve the distributed flowshop scheduling problems. Finally, Section 5 concludes this research work and provides suggestions on future research directions for interested readers.

2. Literature review

Many extensions have been introduced to address case-specific operational situations for broader real-world applications of the flowshop scheduling problem. Distributed flowshop scheduling is a prime example of increasing recognition, motivated by the fact that production planning and scheduling can benefit from advanced communication systems to provide well-informed decisions in distributed manufacturing environments (Ying et al. 2020). Given the complexities involved in managing distributed operations, the distributed scheduling problems (DSP; Jia et al., 2002, 2003) have seen considerable development. This section systematically reviews the published contributions searching the distributed flowshops in the Google Scholar database. Given a total of 93 unique research items, 54 journal papers in the English language are perceived as relevant and considered for a detailed review. The majority of these works, i.e., 42 out of 54, have been published in the past five years, showing that this research topic is experiencing steady growth. We first analyze the review outcomes from a macro perspective concerning; the production setting and the studied performance indicator are considered as suggested by (Neufeld et al. 2016; Ribas et al. 2010). We then dive deeper into the most relevant stream of the research to the present manuscript.

Considering the optimization criterion, 41 articles considered minimizing the maximum completion time (makespan); the rest considered the total completion time (Ali et al. 2020; Wu et al. 2019; Xiong et al. 2014; G. Zhang and Xing 2018), total flow time (Y.-Y. Huang et al. 2021; Q.-K. Pan et al. 2019; Sang et al. 2019; Wu et al. 2018), the weighted sum of completion time (Jeong and Leon 2002), the weighted sum of earliness and tardiness (Jing et

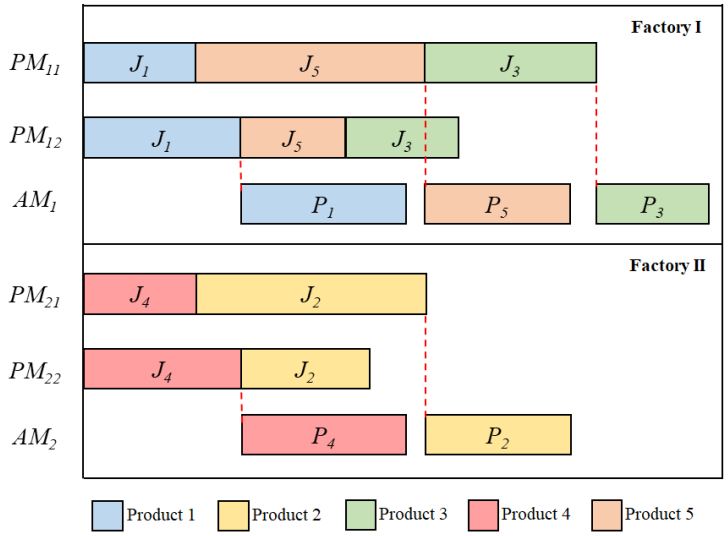
al. 2020), total tardiness (Khare and Agrawal 2020), and total cost of delivery and tardiness (Yang and Xu 2020). Besides, one paper studied a multi-objective variant aiming to minimize maximum and mean completion times (Xiong and Xing 2014).

Distributed scheduling has been extended for various production environments, like distributed flowshop with heterogeneous factories (H. Li et al. 2020; Y. Li, Li, Gao, and Meng 2020; Meng and Pan 2021), group scheduling (Q.-K. Pan et al. 2021), hybrid flowshop (Hao et al. 2019; Lei and Wang 2020; Y. Li, Li, Gao, and Meng 2020; Y. Li, Li, Gao, Zhang, et al. 2020; Ying and Lin 2018), and integrated assembly-production flowshop (Deng et al. 2016; Lei et al. 2020; W.-C. Lin 2018; Wu et al. 2018, 2019; Xiong et al. 2014; Xiong and Xing 2014; G. Zhang and Xing 2018). Various production settings and practical features have also been integrated into the distributed flowshop to facilitate its real-world applications; blocking conditions (W. Li et al. 2019; G. Zhang et al. 2018; Zhao et al. 2020), limited buffer constraints (G. Zhang and Xing 2019), no-wait (Komaki and Malakooti 2017; H. Li et al. 2020; S.-W. Lin and Ying 2016), no-idle (Ying et al. 2017; Zhao et al. 2021), customer order-priority (Meng et al. 2019), time window constraints (Jing et al. 2020), machine-breakdowns (Wang et al. 2016), and preventive maintenance (Mao et al. 2021) are the seminal examples.

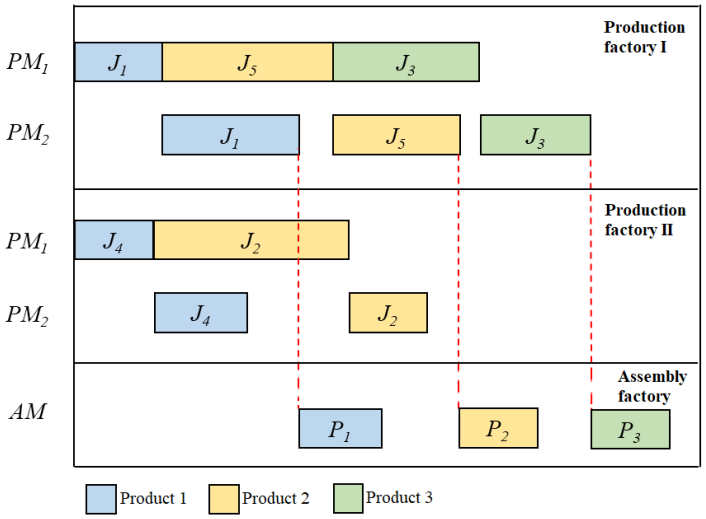
Integrated scheduling of the distributed production-assembly operations has received recent recognition in the scheduling literature. Extending the seminal work of (Naderi and Ruiz 2010), Hatami et al. (2013) introduced the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP) and developed a Variable Neighborhood Descent to minimize the makespan. Later, Hatami et al. (2015) applied the basic Iterated Greedy (IG) algorithm to solve the extended DAPFSP considering setup times and makespan criterion. Other researchers extended these seminal works, for example, by including stochastic processing time within a simulation-based optimization approach

(Gonzalez-Neira et al. 2017), no-idle setting optimized using Water Wave Optimization (Zhao et al. 2021), and a two-stage hybrid flowshop with Shuffled Frog-leaping Optimization (Lei and Wang 2020), all minimizing the makespan. In earlier studies, (J.-Q. Pan et al. 2018) improved the Best-Found Solutions (BFS) considering total flow time. Invasive Weed Optimization of (Sang et al. 2019) addressed the same optimization criterion with the BFSs being later on improved by (Huang et al., 2021). The Genetic Algorithm of (Zhang et al., 2020) is another solution algorithm developed to solve the DAPFSPs, minimizing the makespan.

Yang & Xu (2020) the first time included more than one machine in the assembly plant, the so-called flexible assembly. They also integrated the batch delivery setting into the problem and minimized the total cost of delivery and tardiness. Considering this seminal work, (Ying et al. 2020; G. Zhang et al. 2020) improved the BFS to the DAPFSP with flexible assembly, minimizing the makespan. These studies assumed that the assembly stage is executed in a geographical location different from the production factory and that sub-assemblies are not required after producing the components in the OEM. This assumption does not apply to the production of many products, like fire engines (Lee et al. 1993) and personal computers (Potts et al. 1995), where a two-stage production assembly is required at the OEMs plant. A general example of this situation is when the components should be assembled instantly after the production process because of the operational limitations, for example, immediate assembly after applying sealant/adhesives. Xiong & Xing (2014) extended the Two-Stage Assembly Scheduling Problem (TSASP; Lee et al., 1993) to account for this production setting in a distributed manufacturing environment. The DTSAFSP accounts for integrated production-assembly operations in every production site while the DAPFSP assumes that these operations are conducted in detached places (Figure 1).



(a). DTSAFSP



(b). DAPFSP

Figure 1. A visual illustration of the difference between DTSAFSP and DAPFSP.

Unlike the DAPFSP that has seen considerable development in terms of new mathematical extensions and solution algorithms, the journal papers to DTSAFSPs are limited; apart from the two-stage three-machine assembly scheduling (W.-C. Lin 2018; Wu et al. 2018, 2019), which is a limited edition to the DTSAFSP, only four published works extended the seminal work of (Xiong and Xing 2014). Xiong et al. (2014) developed a Variable Neighborhood Search algorithm to minimize the total completion time in DTSAFSP with setup times. G. Zhang & Xing (2018) developed a Social Spider Optimization algorithm to minimize total completion time in DTSAFSP. Deng et al. (2016) developed a Competitive Memetic

Algorithm (CMA) to introduce the best-found makespan value to the DTSAFSPs, which outperformed the VNS algorithm and the hybrid of GA with RVNS developed by Xiong et al. (2014). The Cooperated Teaching-Learning-based Optimization (Lei et al. 2020) is more recent research that failed to outperform the CMA algorithm in terms of solution quality and computational time. No scheduling studies accounted for a mixed-setup situation; this operational situation is prevalent when different operations or production technologies in different factories are considered. The present study proposes a new mathematical formulation and a novel solution algorithm, the MIG algorithm, to solve this emerging scheduling extension comparing it with the best-performing algorithm, CMA.

3. Proposed method

3.1. Mathematical formulation

Let assume f homogeneous factories producing products in an SC. Each of the factories includes both production and assembly operations and is equipped with machinery to process a total of $N = \{0, 1, 2, ..., n\}$ jobs/products. The machines in the production stage of the factories are multi-purpose; hence, require setup times between two consecutive assignments. However, the assembly stage does not require a setup due to the use of automated assembly systems. Jobs follow the same routine across the SC. It is assumed that each job $j \in N$ can be processed on exactly one machine/stage at a given time. Moreover, setup times are assumed to be job sequence-dependent at the production stage while the assembly stage does not require setup operations. The processing times are deterministic but independent of the order of the job/products. It is also assumed that the assignments are executable immediately at the starting point of the planning period. Once assigned, the jobs can no longer be re-assigned to another factory. The model is hereafter denoted by

$(DPm \to 1) \to 0 \,|\, mixed - ST_{sd} \,|\, C_{max}$, where $(DPm \to 1) \to 0$ indicate that the distributed system has m parallel machines in the production stage of each factory and one assembly machine completing the final assembly operations; $mixed - ST_{sd}$ specifies that the production stage is characterized by SDSTs, while the assembly operations do not require a setup; the maximum completion time (makespan) of the assembly operations is denoted by C_{max} indicating the optimization criterion.

Notations

n	The total number of jobs.
m	The total number of available machines in the production stage.
f	The number of factories in the SC.
i , j	Job index, where $i, j \in \{0, 1, 2, ..., n\}$ with 0 being a dummy job.
k	Machine tag, $k \in \{1, 2, ..., m\}$.
$S_{i,j,k}$	Setup time at the production stage indicating the preparation required for processing job j after job i on machine k.
$P_{j,k}$	The processing time required to complete job i on machine k at the production stage.
AP_j	The processing time required for the assembly of product j.
$X_{i,j}$	Binary decision variable; $=1$ if job j is processed after job i; $=0$, otherwise.
$C_{j,k}$	Integer decision variable indicating the completion time of job j on machine k at the production stage.
AC_j	Integer decision variable indicating the completion time of job j at the assembly stage

The Mixed-Integer Linear Programming (MILP) formulation is provided below.

$$Min \quad C_{\max} \tag{1}$$

Subject to:

$$\sum_{i=0, i\neq j}^{n} X_{i,j} = 1 \quad \forall j \in \{1,2,...,n\} \tag{2}$$

$$\sum_{j=0}^{n} X_{i,j} \leq 1 \quad \forall i \in \{1,2,...,n\} \tag{3}$$

$$\sum_{j=1}^{n} X_{0,j} = f \tag{4}$$

$$\sum_{i=1}^{n} X_{i,0} = f - 1 \tag{5}$$

$$X_{i,j} + X_{j,i} \leq 1 \; ; \; i \in \{1,2,...,n-1\}, \; j > i \tag{6}$$

$$C_{j,k} \geq C_{i,k} + S_{i,j,k} + P_{j,k} - M \times \left(1 - X_{i,j}\right) \quad \forall i \in \{0,1,...,n\}, j \neq i, \; k \in \{1,2,...,m\} \tag{7}$$

$$AC_j \geq C_{j,k} + AP_j - M \times (1 - X_{i,j}) \quad \forall j \in \{1,2,...,n\}, k \in \{1,2,...,m\} \tag{8}$$

$$AC_j \geq AC_i + AP_j - M \times (1 - X_{i,j}) \quad \forall i \in \{0,1,...,n\}, \; j > i, \; k \in \{1,2,...,m\} \tag{9}$$

$$C_{\max} \geq AC_j \quad \forall j \in \{1,2,...,n\} \tag{10}$$

$$C_{0,k} = AC_0 = 0 \quad k \in \{1,2,...,m\} \tag{11}$$

$$\mathrm{X}_{i,j} \in \{0,1\} \quad \forall i \in \{0,1,...,n\}, j \in \{0,1,...,n\}$$

$$\mathrm{C}_{j,k} \geq 0 \quad \forall j \in \{1,2,...,n\}, k \in \{1,2,...,m\} \tag{12}$$

$$AC_j \geq 0 \quad \forall j \in \{0,1,...,n\}$$

The objective function in Equation (1) minimized the maximum completion time. Constraints (2) and (3) use binary variables to ensure that the job assignments are unique and that there is only one job before and after every job. Equations (4) and (5) ensure that the job

sequence positioned before and after the virtual job in every factory, respectively, appear exactly f times. According to Constraint (6), a job cannot be arranged before and after another job simultaneously. M is defined as a large positive number to establish a connection between the decision variables. On this basis, Constraints (7) indicate that if the job j is located immediately after the job i, its completion time is greater than the summation of its processing time, the completion time of the preceding job, the associated setup time in the production stage. Constraint set (8) specifies the assembly procedure takes place only after completing the processing of the components in the production stage. Constraint (9) ensures that if the product j is assembled after the product i, the assembly completion time is greater than the summation of its assembly processing time and the assembly completion time of the product i; this also ensures that the sequence of product assembly on the same assembly machine in the factory is preserved. Constraints (10) define the objective function as the maximum completion time of all products on the assembly stage. Equation (11) specifies that the completion time of the auxiliary job on every machine is set equal to zero. Finally, the last constraint determines the type of decision variables.

3.2. Solution algorithm

The IG algorithm cannot effectively avoid the local optimality traps in the complex solution spaces (Ying et al. 2020). Case-specific methods have been proposed to address this shortcoming in various flowshop environments, like hybrid flowshop (Shao et al. 2020), hybrid flexible flowshop (Ozsoydan and Sagir 2021), blocking flowshop (Cheng et al. 2021), and no-idle permutation flowshop (Riahi et al. 2020), among others. Considering the nature of the scheduling problems, in particular, the distributed production-assembly setting and mixed-setup constraint, this study develops the MIG algorithm, which is fortified by the Meta-Lamarckian Learning (MLL)-based perturbation mechanism to improve the exploration

power of the metaheuristic with more effectively evading the local optimality traps. The pseudocode of the MIG algorithm is provided in Figure 2. The computational procedure consisting of the solution initialization, destruction & construction, the MLL-based local search algorithm, and the acceptance mechanisms, are explained as follows.

```
1  procedure  MIG(d)
2        π = NEH_2_heuristic ;                  // Solution Initialization
3        π_best = π;                            // Best Solution Storage
4        d = Destruction _ count
5        π' = π;                                // Temporary Storage
6           for i = 1 to d do                   // Destruction procedure
7              π' = remove a randomly selected job
8           endfor // π' is the partial solution;
9           for i = 1 to d do                   // Construction procedure
10             π' = insert the job into π';
11          endfor
12          π'' = Training_phase(π);           //MLL _based Local Search
13          π'' = Working_phase(π');
14          if C_max(π'') < C_max(π) then      // Acceptance Mechanism
15             π = π'';
16             if C_max(π'') < C_max(π') then
17                π' = π'';
18             endif
19          else if (random ≤ exp(−RPD)) then
20             π = π'';
21          endif
22       endif
23       return π
24 end
```

Figure 2. Pseudocode of the Meta-Lamarckian-based Iterated Greedy (MIG) Algorithm.

3.2.1. Initialization/encoding module

The solution representation method displays the job assignments to all the factories in a single vector. Considering an illustrative example with five jobs and two factories, a solution can be

represented by $\pi = \{3\ 5\ 1\ 0\ 2\ 4\}$, where 3 and 2 jobs are assigned to the first and second factories, respectively, and 0 is used as a delimiter to separate the factory assignments. Given this encoding mechanism, one should initialize solutions that are feasible and acceptably good. The basic IG algorithms apply random initialization methods, which are ineffective for complex problems. The following initialization procedure is tailored for distributed scheduling.

Table 1 provides the details of the above illustrative example to explain the procedure better. The first and second parts of the table provide the processing and sequence-dependent setup times (SDSTs) in the production and assembly stages, respectively, to account for a more generalized setting. The initialization procedure begins with calculating the completion time of every job at the assembly stage as an individual. That is, $AC_j = \max_{\forall k}\{S_{0,j,k} + P_{j,k}\} + AP_j$. In doing so, the completion time of jobs 1-5 is equal to 15, 16, 18, 19, and 14, respectively. Then, selecting the job associated with the shortest completion time, jobs 5 and 1 are assigned to factories 1 and 2, respectively. Next, job 2 is added next to jobs 5 and 1 in factories 1 and 2, respectively. The partial solution $\pi_0' = \{\ 5\ 2\ 0\ 1\ \}$ appeared to be a better option at this stage due to a smaller makespan value, 25. Next, the remainder of unallocated jobs should be added into the partial solution after trying all the possible positions in the vector. For this purpose, job 4 is inserted into all positions in every section of the vector to exhaust all possible options and calculate the fitness values. Inserting job 4 next to job 1 in factory 2 is found to be the best option with a makespan value of 28. Applying the same approach, job 3 appeared to be at its best position when it is located next to job 2 in factory 1, resulting in a makespan of 39. Therefore, the completed sequence $\pi_0 = \{\ 5\ 2\ 3\ 0\ 1\ 4\ \}$ should be selected as the initial solution.

Table 1. An illustrative example for the computational procedure of the MIG algorithm.

Processing time	*Job*				
Machine	1	2	3	4	5
PM_1	6	8	9	7	4
PM_2	6	3	5	6	7
AM	5	6	7	7	5

SDSTs on PM_1/PM_2	*Job*					
Job	*0*	*1*	*2*	*3*	*4*	*5*
0	0/0	3/4	2/4	2/5	1/6	3/2
1	0/0	0/0	4/5	4/4	5/1	3/7
2	0/0	4/4	0/0	4/3	6/3	7/1
3	0/0	1/1	3/6	0/0	1/2	2/4
4	0/0	4/3	2/4	1/7	0/0	4/2
5	0/0	4/2	4/3	3/5	4/4	0/0

3.2.2. Destruction module

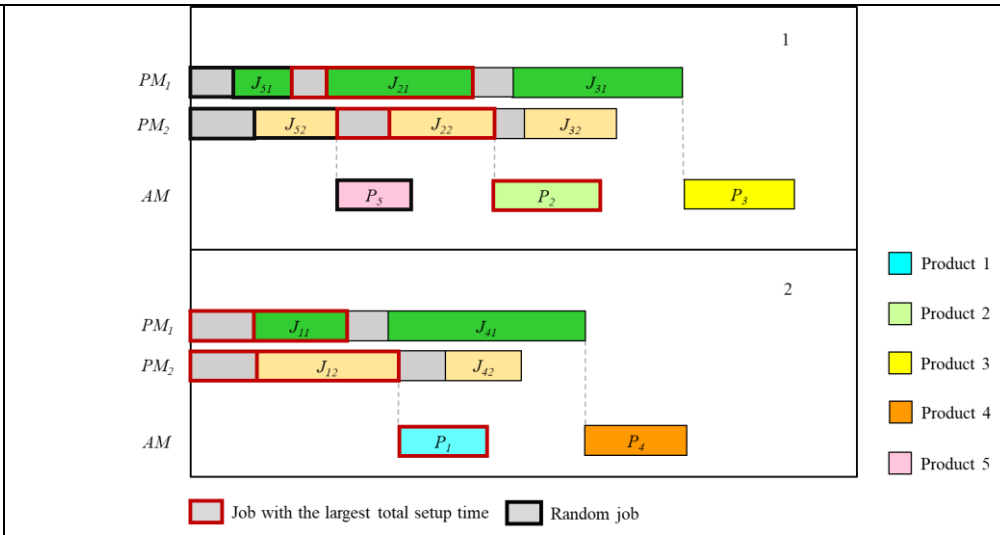
The destruction and construction procedures are at the heart of the IG computations, where an input solution to the module will be first destructed up to a certain point, and an iterative procedure is then applied to reconstruct a new complete solution. This research customizes the destruction method of (Ruiz et al. 2019), which was developed for distributed permutation flowshop scheduling. Given the destruction count, d, the method begins with selecting $\lceil d/2 \rceil$ jobs with the largest total setup time from the factory with the largest C_{max}, which is hereafter labeled by F_c. Next, the rest of the $1-\lfloor d/2 \rfloor$ jobs in the remaining $F-1$ factories are extracted following the same approach. The removed jobs are then saved in a separate vector considering the same order they are extracted.

Considering the illustrative example in Table 1, where $\pi_0 = \{ 5\ 2\ 3\ 0\ 1\ 4 \}$ is the input solution, the maximum completion time of factories 1 and 2 are $C_{max} = 39$ and $C_{max} = 28$, hence, F_c is 1. Given a destruction count of $d = 3$, $\lceil 3/2 \rceil = 2$ jobs should be removed

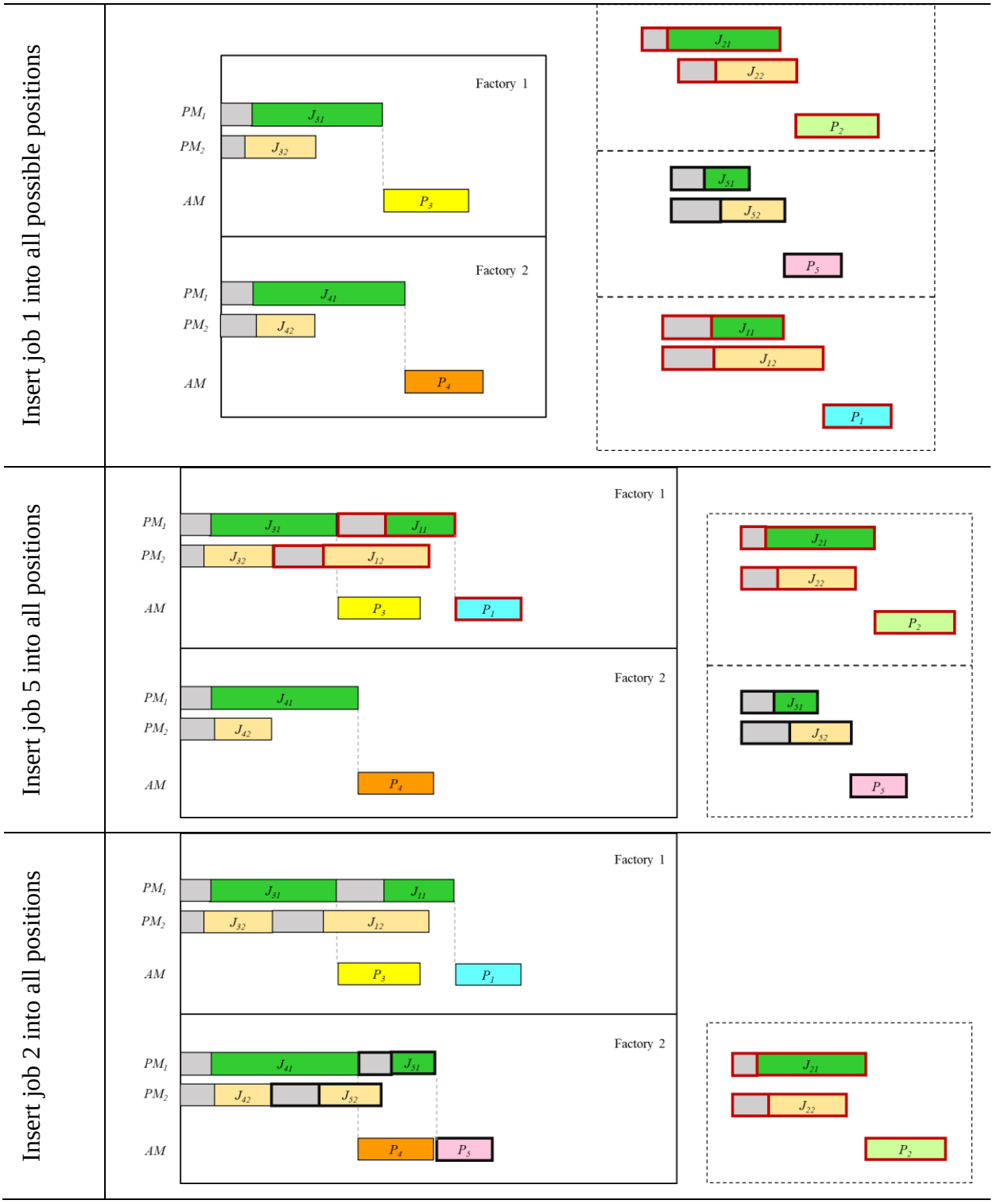
from factory 1. For this purpose, the job associated with the largest total setup times in the production phase of factory 1, i.e., job 2, and a random job, let assume job 5, are removed first. Next, 1-$\lfloor 3/2 \rfloor = 1$ jobs are selected from the remaining factories, factory 2 in this example, with the target being the largest total setup time, i.e., job 1. In so doing, the list of removed jobs in the destruction phase is $\pi_d = \{ 2\ 5\ 1 \}$, and $\pi' = \{ 3\ 0\ 4 \}$ is the partial solution with $n-d$ jobs. This procedure is illustrated in the destruction phase of Figure 3. In this example, *J* refers to the jobs (parts) and *P* indicates the products that are an assembly of several jobs (parts). On this basis, the first index of *J* corresponds to the index of *P* (product number) while the second index of *J* determines the component number required for assembling the corresponding product.

Destruction Phase: Extract *d*=3 jobs from across factories

Remove two jobs from factory 1 and 1 job from factory 2



Construction Phase: insert the extracted jobs into the partial solution in a reverse order


Insert job 1 into all possible positions
Factory 1
Factory 2
PM1
PM2
AM
J31
J32
P3
J41
J42
P4
J21
J22
P2
J51
J52
P5
J11
J12
P1
Insert job 5 into all positions
Factory 1
Factory 2
PM1
PM2
AM
J31
J11
J32
J12
P3
P1
J41
J42
P4
J21
J22
P2
J51
J52
P5
Insert job 2 into all positions
Factory 1
Factory 2
PM1
PM2
AM
J31
J11
J32
J12
P3
P1
J41
J51
J42
J52
P4
P5
J21
J22
P2

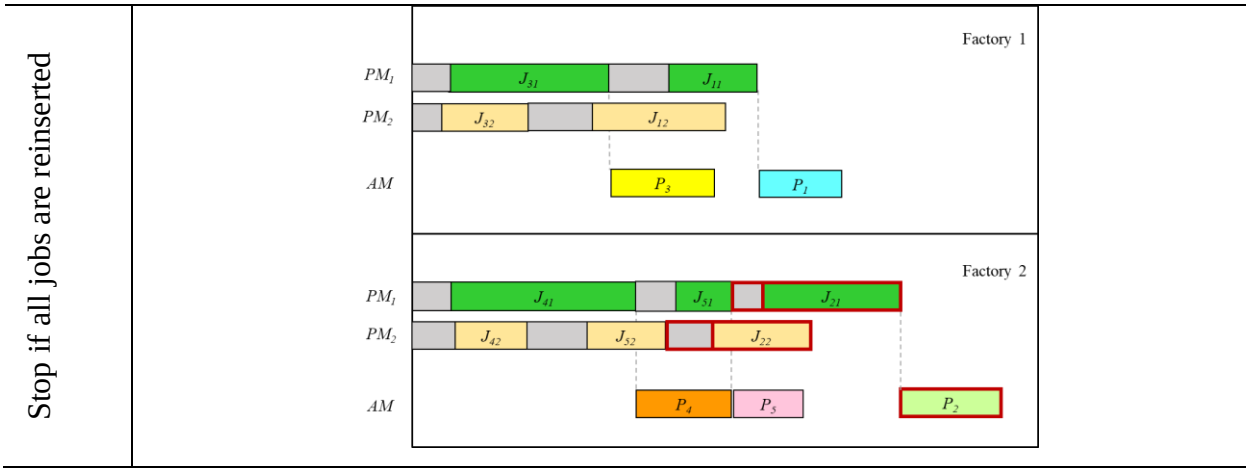


Figure 3. A visual illustration of the destruction and construction procedures.

3.2.3. Construction module

In the construction phase, the deleted jobs from π_d are to be reinserted into all possible positions in the partial solution, π'; this is done in reverse order of extraction in the destruction phase. For this purpose, the insertion resulting in the smallest makespan will dominate the greedy search at every step. After inserting a job into the best position, the jobs immediately before and after the newly inserted job will be displaced into all possible positions across factories, and the option with the smallest makespan will be selected.

Assuming that job 1 is extracted in order from π_d and inserted into all possible positions in π', the alternative $\{3\ 1\ 0\ 4\}$ results in the minimum completion time. In this situation, the jobs before and after are 3 and 0; the delimiter that should not be extracted. Inserting 3 into all possible positions across factories, $\pi' = \{3\ 1\ 0\ 4\}$ records the smallest makespan value. The list of unassigned jobs is now updated to $\{5\ 2\}$. Next, one needs to find the best insertion position for job 5, which is $\pi' = \{3\ 1\ 0\ 4\ 5\}$, and displace the preceding job, job 4, into the best position, which is $\pi' = \{3\ 1\ 0\ 4\ 5\}$. Given $\{2\}$ as the last unassigned job, the trial and error for finding the best insertion follows the same procedure that results in

$\pi' = \{ 3\ 1\ 0\ 4\ 2\ 5 \}$. As the last step, the jobs before and after job 2, namely jobs 4 and 5, are now extracted in order and inserted into all possible positions; $\pi = \{ 3\ 1\ 0\ 2\ 4\ 5 \}$ is the best found alternative; given that the list of unassigned jobs is now empty, the resulting job sequence is a complete solution and can be considered for the next step of computations, i.e., the local search moves. This procedure is demonstrated in the construction phase of Figure 3.

3.2.4. MLL-based local search algorithm

The MLL-based mechanism (Ong and Keane 2004) was introduced to improve the quality of the best individual in the solution population, enhancing the effectiveness of the Memetic Algorithms. G. Zhang et al. (2020) showed that the novel perturbation mechanism not only effectively avoids falling into local optimality traps but also is comparatively more efficient than the other hybrid approaches. Inspired by its positive implications, our study extends to integrate the MLL-based approach into the IG algorithm for solving a new scheduling extension. Four basic and two hybrid local search moves, shown in Figures 4-5, are defined below to formally introduce the MLL-based algorithm.

- Job Swap (JS) selects a random factory to exchange two random jobs. Assuming $\pi = \{ 3\ 1\ 0\ 2\ 4\ 5 \}$ as the job sequence after the construction phase and $F_r = \{ 2\ 4\ 5 \}$ being the randomly selected factory, J4 and J5 are swapped as random jobs to obtain an alternative solution, $\pi'_{JS} = \{ 3\ 1\ 0\ 2\ 5\ 4 \}$.

- Job Competitive Insertion (JCI), a random job is first removed from a random factory; it is then inserted into all possible positions of the same factory to find the best alternative. Assuming $\pi = \{ 3\ 1\ 0\ 2\ 4\ 5 \}$ and J5 of factory 2 as our random selection, the resulting alternatives are $\pi'_{JCI} = \{ 3\ 1\ 0\ 5\ 2\ 4 \}$ and

$\pi'_{JCI} = \{\ 3\ 1\ 0\ 2\ 5\ 4\ \}$.

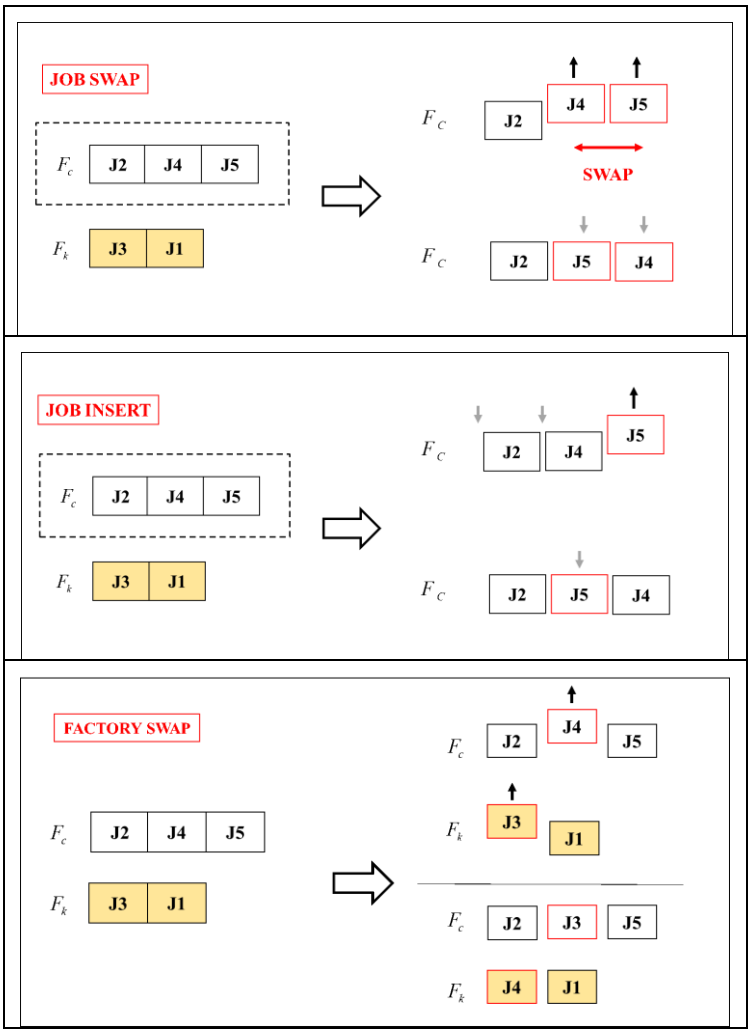
- Inter-Factory Swap (IS) move considers two randomly selected factories, F_c and F_k, and selects a random job from each. It then exchanges the position of the selected jobs. For instance, considering J3 and J4 as the random jobs from the first and the second factories, $\pi'_{IS} = \{\ 4\ 1\ 0\ 2\ 3\ 5\ \}$ is the neighboring solution.

- Inter-Factory Competitive Insertion (ICI) removes a random job from the first random factory, F_c, and inserts it into all possible positions in the second random factory, F_k. For example, assuming $F_c = \{\ 2\ 4\ 5\ \}$, $F_k = \{\ 3\ 1\ \}$ and J4 as the random selections, exhausting the insertion options results in $\pi'_{ICI} = \{\ 2\ 5\ 0\ 4\ 3\ 1\ \}$, $\pi'_{ICI} = \{\ 2\ 5\ 0\ 3\ 4\ 1\ \}$, and $\pi'_{ICI} = \{\ 2\ 5\ 0\ 3\ 1\ 4\ \}$, among which the best should be selected. Finally, Inter-Factory Competitive Insertion consists of displacing the sequence assigned to one factory before and after every other factory to find a better alternative.

- The Hybrid Swap (HS); JS and IS are executed simultaneously, illustrated in Figure 5(a), where (a). two random jobs are swapped in the first randomly selected factory, $F_c = \{\ 2\ 4\ 5\ \}$, and, immediately, (b). one job is extracted from the modified F_c and a random job from the second randomly selected factory, $F_k = \{\ 3\ 1\ \}$, for a second exchange. At this point, the perturbation will be accepted only if it results in better fitness values, makespan in this study. Assuming J2 and J5 for step *a* as well as J4 and J5 for step *b*, $\pi' = \{\ 3\ 4\ 0\ 5\ 1\ 2\}$ is resulted.

- Hybrid Competitive Insertion (HCI) executes JCI and ICI without interruption, as shown in Figure 5(b). That is, two jobs are extracted from the randomly selected

factory, F_c. The following procedure comes next; (1). The first insertion exhausts all the possibilities in the same factory considering the first random job; (2). The second randomly selected job will be inserted into all possible positions in the other randomly selected factory, F_k. A neighboring solution will be selected only if it is a breakthrough. Assuming J5 and J4 are selected in step 1 of the illustrative example, J5 is first inserted into all possible positions in the same factory, resulting in $\{3\,1\,0\,5\,2\}$. Then, J4 is inserted into all possible positions in the second factory. This procedure finds $\pi' = \{4\,3\,1\,0\,5\,2\}$ as the best alternative.



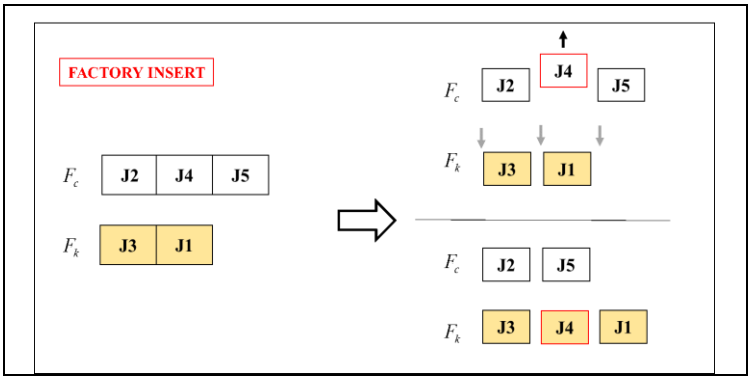


Figure 4. A visual illustration of the basic local search moves.

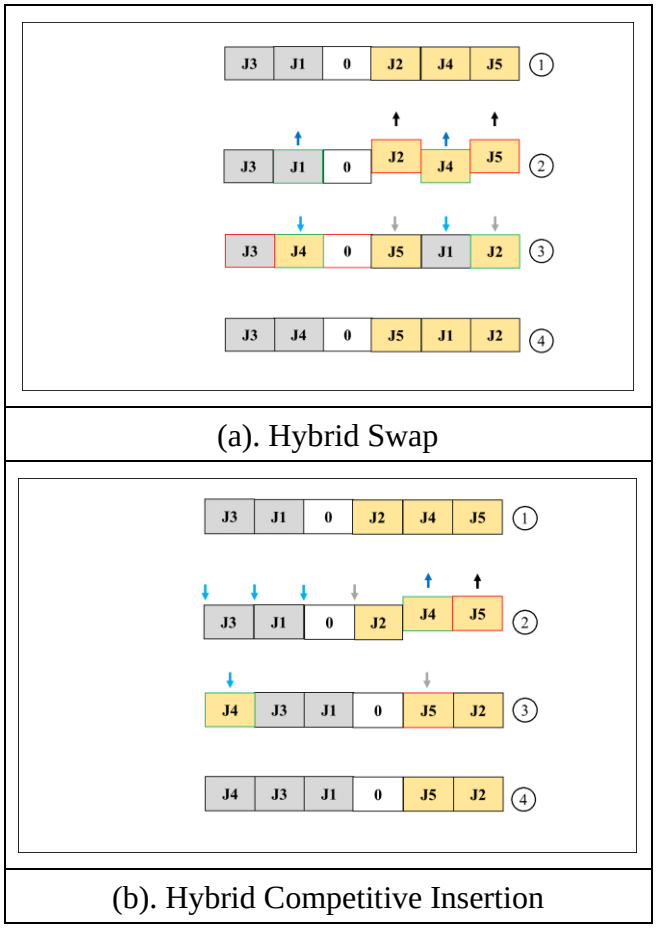


Figure 5. A visual illustration of the hybrid local search moves.

The MLL-based mechanism consists of the training and implementation phases. Based on the study of Zhang et al. (2020), the HS and HCI are executed $n\times(n-1)$ times on the initial solution, respectively, in the training phase. Then, the fitness function values $\eta_k\,(k=1,\,2)$ of HS and HCI can be calculated using Equation (13), where C_b denotes the makespan of the solution before executing the HS/HCI, and C_a represents the makespan of the best solution

after executing the HS/HCI $n\times(n-1)$ times. On this basis, the selection probability values $\zeta_k\,(k=1,\,2)$ of HS and HCI should then be calculated using Equation (14) to conclude the training phase.

$$\eta_k = \frac{(C_b - C_a)}{n\times(n\ -1)} \tag{13}$$

$$\zeta_k = \frac{\eta_k}{\sum_{i=1}^{2}\eta_i} \tag{14}$$

Considering the best results from the non-initial solution population, the implementation phase applies the JS move to perturb with $\pi=\{4\ 3\ 1\ 0\ 5\ 2\}$ being the result. The implementation procedure continues by applying the roulette wheel selection method to select from $\{HS, HCI\}$. The selected hybrid approach should then be applied $n\times(n\ -1)$ times. The resulting solutions are appraised by calculating the fitness value, makespan in this study, to check whether or not the overall performance is improved. It is worthwhile noting that the reward and selection probability values should be updated after each move, that is, $\eta_k \leftarrow \eta_k + \eta_{new}$. This procedure should be repeated for the JCI, IS, and ICI local search moves.

3.2.5. Conditional mechanisms

Two conditional mechanisms are considered: the acceptance mechanism and the termination criterion. The former decides whether to accept a new solution, and the latter determines when to stop the search procedure.

Traditionally, a new solution is accepted only if it performs strictly better than the old solution. However, this approach is prone to missing out on promising solutions when searching complex solution spaces. Hatami et al. (2015) introduced a competitive acceptance

criterion to help reduce the odds of falling within local optimality, addressing the mentioned shortcoming. On this basis, every new solution will be evaluated after passing the construction and local search procedures. This study adopts the acceptance mechanism introduced by (Hatami et al. 2015) to help improve the performance of the search algorithm. In this approach, the acceptance probability is based on Expression (15), where Relative Percentage Difference (RPD) is the comparative difference between the new and old solutions, which is calculated using Equation (16). In this definition, the smaller the RPD value, the better the solution quality and the higher the likelihood of accepting it. A random number between zero and one is first generated whenever the acceptance function is provoked. If the random number is smaller than $e^{-\mathrm{RPD}}$, the new solution should be accepted. Otherwise, the new solution will be discarded from the memory.

$$\text{random} \leq e^{-\mathrm{RPD}} \tag{15}$$

$$RPD = \frac{C_{\max}(\pi') - C_{\max}(\pi)}{C_{\max}(\pi)} \times 100 \tag{16}$$

The maximum CPU time and the maximum iterations are the most widely applied termination condition in the scheduling literature (Cheng et al. 2021); the former is considered as the termination criterion of the MIG and the benchmark algorithms to ensure a fair analysis of the algorithms' performance. Therefore, the algorithms terminate and return the (near-) optimum solutions when CPU time equals $0.1 \times n$, where the threshold increases proportionate to the workload. The resulting fitness values are analyzed in the numerical analysis section.

4. Numerical analysis

This section analyzes the results of the extensive numerical experiments for evaluating the performance of the developed solution algorithm, MIG. To proceed with the experiments, the

MIG algorithm is first calibrated using 120 random test instances, and the Relative Performance Deviation (RPD) measure is calculated using Equation (16). The smaller RPDs are preferred with $C_{\max}(\pi)$ and $C_{\max}(\pi')$ specifying the best-found makespan and the makespan under consideration, respectively. The calibration results are summarized in Table 2, based on which a destruction value equal to 4 is considered for conducting the final experiments. The remainder of this section begins with a description of the configuration of the test instances. Results analysis and statistical test of significance follow to evaluate the developed solution method. The benchmark algorithms, MIG and CMA, are coded and compiled using C++ programming language on a personal computer with the following specs: Intel® Core™ i7-7700 CPU 3.60GHz, 16GB RAM, and Windows 10 Operating System. Besides, Gurobi 9.0.3 is used to confirm the correctness of the developed mathematical formulation and compared with the results obtained by the metaheuristic algorithms for small test instances.

Table 2. Test results for calibration of the MIG algorithm (best result in **bold**).

Destruction Count	*Relative Performance Deviation*
2	2.686
3	2.445
4	**1.634**
5	1.853
6	1.676
7	1.690
8	1.784

4.1. Dataset description

This study follows Deng et al. (2016) to configure a new set of test instances. On this basis, various numbers of jobs $\{10, 20, 50, 100, 200, 500\}$, machines $\{2, 4, 6, 8\}$, and factories $\{2, 3, 4, 5, 6\}$ are considered to analyze the impact of operational parameters on

the scheduling outcomes. The processing and setup times are generated randomly using a continuous uniform distribution function with $U(1, 100)$ and $U(1, 20)$ seconds, respectively. Considering ten variants to each of $f \times n \times m = 5 \times 6 \times 4 = 120$ configurations, a total of 1200 test instances are considered for the numerical experiments. The solution algorithms are run ten times for solving each of the test instances; the results are provided below.

4.2. Results analysis

First, test instances with a small workload of 10 jobs are solved using the exact solution approach (MILP), CMA, and the MIG algorithm. The results are compared considering RPD and summarized in Table 3 with 'zero' indicating the best-obtained results. This table also reports the number of successful attempts in obtaining the best results and the required calculation time. It is observed that CMA performs relatively steadier in solving very small test instances. Expectedly, the computational time of the MIG and CMA algorithms for solving very small instances are significantly shorter than that of the exact algorithm, while the best solutions found by these metaheuristics are close to the exact value.

Table 3. Exact solution results analysis over very small test instances.

Parameter		MIG				CMA				MILP			
		Best	RPD	No.	Time	Best	RPD	No.	Time	Best	RPD	No.	Time
Factories *(f)*	2	332.67	2.14	16	1.005	330.45	1.22	30	1.000	329.97	0.00	40	307.859
	3	237.05	2.45	29	1.006	236.12	1.15	37	1.000	235.77	0.00	40	32.258
	4	188.07	3.01	31	1.009	187.37	0.76	38	1.000	187.25	0.00	40	5.895
	5	162.07	3.63	22	1.013	158.95	0.56	40	1.000	158.95	0.00	40	0.527
	6	147.25	3.07	37	1.011	146.77	0.31	40	1.000	146.77	0.00	40	0.327
Machines *(m)*	2	190.44	3.38	29	1.012	188.66	0.96	44	1.000	188.44	0.00	50	38.001
	4	210.22	2.87	34	1.008	208.50	0.86	48	1.000	208.42	0.00	50	49.669
	6	222.92	2.96	32	1.008	220.88	0.73	47	1.000	220.72	0.00	50	74.181
	8	230.12	2.24	40	1.008	229.70	0.66	46	1.000	229.40	0.00	50	115.642

Next, the BFS to each of the 1200 test instances is considered for macro analysis. Analyzing the results considering various workloads, the number of machines in each factory, and the number of factories are summarized in Table 4. The overall analysis of the results shows that CMA performs slightly better for solving problems with small workloads, i.e., 10 and 20 jobs. However, better solutions are yielded by MIG for instances with more than 50 jobs. The over-performance gap between MIG and CMA increases sharply when solving larger test instances with about a 5 percent difference for solving the problems with 500 jobs. One can expect that the difference becomes even more significant when addressing very large-scale industrial cases. Besides, the results obtained by MIG are notably steadier for solving medium- and large-scale problems with standard deviations of around 10 percent of that resulting from CMA. Analyzing the impact of operational scale, i.e., an increase in the number of machines and factories, shows that MIG yields better solutions in all the categories. It indicates that the superiority of MIG over CMA in solving large-scale instances is more significant than that of the results obtained by these algorithms in solving small-scale instances.

Table 4. Results analysis considering various operational parameters (best in **bold**).

Parameter		CMA			MIG		
		Best	Ave	StD	Best	Ave	StD
Jobs *(n)*	10	**211.94**	**213.60**	**1.66**	213.42	217.35	3.13
	20	**378.58**	**384.90**	**3.78**	380.96	389.61	5.33
	50	**886.69**	**897.59**	6.84	888.32	898.71	**6.29**
	100	1751.22	1769.73	11.64	**1737.92**	**1750.08**	**7.47**
	200	3491.11	3517.71	17.41	**3417.78**	**3430.57**	**7.63**
	500	8822.22	8984.95	159.19	**8417.00**	**8429.51**	**7.23**
Machines *(m)*	2	2492.77	2513.30	15.30	**2394.21**	**2403.05**	**5.39**
	4	2582.57	2606.16	19.28	**2503.52**	**2513.54**	**6.34**
	6	2626.64	2671.05	43.03	**2553.39**	**2563.84**	**6.19**
	8	2659.20	2721.81	56.06	**2585.83**	**2596.79**	**6.81**
Factories *(f)*	2	4407.70	4428.94	14.06	**4270.33**	**4284.04**	**8.06**
	3	2964.15	2997.67	36.80	**2875.87**	**2886.77**	**6.57**
	4	2244.45	2285.03	36.50	**2176.07**	**2185.73**	**6.06**
	5	1809.51	1853.60	38.50	**1752.56**	**1760.65**	**5.08**
	6	1525.65	1575.16	41.25	**1471.34**	**1479.33**	**5.15**

The extent of difference between the performances of the benchmark algorithms can be better observed by analyzing the Average Relative Performance Deviation (ARPD) values. According to Table 5, the difference in the results for the instances with 500 jobs is 0.15 (MIG) to 7.43 (CMA), which showcases the superiority of MIG over the best-performing solution algorithm in the literature, CMA, for solving industry-scale problems. The gap becomes smaller with a decrease in the workload. In contrast, the impact of the change in the number of machines and factories appeared to be negligible because the difference between ARPDs resulting from MIG and CMA remains about the same over various parameter values. Notably, MIG shows a higher accuracy considering the number of successful attempts in search for the BFS.

Table 5. Analysis of the Relative Performance Deviation (RPD) (best in **bold**).

Parameter		CMA		MIG	
		RPD	Successful Attempts	RPD	Successful Attempts
Jobs (*n*)	10	**0.80**	**185**	2.86	135
	20	**2.12**	**146**	3.63	71
	50	**1.54**	**117**	1.71	95
	100	1.98	20	**0.78**	**182**
	200	3.04	0	**0.39**	**200**
	500	7.43	0	**0.15**	**200**
Machines (*m*)	2	2.88	117	**1.58**	**216**
	4	2.55	116	**1.56**	**224**
	6	2.81	112	**1.63**	**223**
	8	3.04	123	**1.59**	**220**
Factories (*f*)	2	2.17	76	**0.99**	**175**
	3	2.40	97	**1.27**	**176**
	4	2.79	92	**1.66**	**183**
	5	3.09	101	**1.90**	**167**
	6	3.64	102	**2.11**	**182**

As a final step to our analysis, a paired *t*-test is conducted to check whether there is a

statistical difference between the means of two groups of fitness values considering 1200 test instances. Considering a confidence level of 95 percent, the null hypothesis speculates that the two population means are equal, that is, the difference between the two groups of fitness values is equal to 0. The results reported in Table 6 indicate that *t*-Statistic value is greater than *t*-Critical. With the *p*-value being less than the significance level, we can reject the null hypothesis. It can be confirmed that the difference between the results obtained by the MIG, and CMA algorithms is significant and that MIG can be now regarded as the best-performing algorithm for optimizing the DTSAFSP-MS variant of DSPs.

Table 6. Paired *t*-test analysis of the performance difference between MIG and CMA.

Mean	StD	Error	DoF	*t*-Statistic	*t*-Critical	*p*-value
0.0123	0.0362	0.0010	1199	11.7886	1.9619	0.0000

5. Conclusions

Integrating production and assembly operations in distributed manufacturing environments is a notable evolution in the scheduling literature; it helps optimize SCs and has implications for lean manufacturing by maximizing productivity within an integrated planning procedure. This study provides a comprehensive review of the distributed two-stage flowshop scheduling literature, a new mathematical formulation, and a novel extension to the IG algorithm as its major contributions. An extensive numerical analysis with 1200 test instances shows that the MIG algorithm yielded the vast majority of the BFSs. The statistical test confirmed the significance of the resulting improvement, confirming MIG as a strong benchmark algorithm for optimizing the DTSAFSP-MS.

This study can be extended in several directions. First, two-stage assembly production can be combined with the flexible assembly in a separate factory to simultaneously account for

sub-assemblies at OEMs and the final assembly at the mother company. Second, this research is limited in that the operational parameters are assumed to be deterministic. The next direction for future research can address this shortcoming through stochastic optimization and/or the use of fuzzy and gray-based methods. Third, the mixed-setup production setting can be tested in other scheduling environments with heterogeneous production technology and operations to address this basic assumption. Fourth, additional job- and process-related constraints can be included in the DTSAFSP-MS formulation to account for case-specific operational requirements. Finally, we feel that new approximation algorithms can be developed to improve further the BFS found in our study. For this purpose, applications of supervised- and unsupervised-learning-based methods are worthwhile research topics to address less-tangible aspects of the operations.

References

Ali, A., Gajpal, Y., & Elmekkawy, T. Y. (2020). Distributed permutation flowshop scheduling problem with total completion time objective. *OPSEARCH*, 1–23. https://doi.org/10.1007/s12597-020-00484-3

Cheng, C.-Y., Pourhejazy, P., Ying, K.-C., & Huang, S.-Y. (2021). New Benchmark Algorithm for Minimizing Total Completion Time in blocking flowshops with sequence-dependent setup times. *Applied Soft Computing*, 107229. https://doi.org/10.1016/j.asoc.2021.107229

Deng, J., Wang, L., Wang, S., & Zheng, X. (2016). A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem. *International Journal of Production Research*, *54*(12), 3561–3577. https://doi.org/10.1080/00207543.2015.1084063

Fazel Zarandi, M. H., Sadat Asl, A. A., Sotudian, S., & Castillo, O. (2020). A state of the art review of intelligent scheduling. *Artificial Intelligence Review*, *53*(1), 501–593. https://doi.org/10.1007/s10462-018-9667-6

Gonzalez-Neira, E. M., Ferone, D., Hatami, S., & Juan, A. A. (2017). A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, *79*, 23–36. https://doi.org/10.1016/j.simpat.2017.09.001

Hao, J.-H., Li, J.-Q., Du, Y., Song, M.-X., Duan, P., & Zhang, Y.-Y. (2019). Solving distributed hybrid flowshop scheduling problems by a hybrid brain storm optimization algorithm. *Ieee Access*, *7*, 66879–66894. https://doi.org/10.1109/ACCESS.2019.2917273

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2013). The Distributed Assembly Permutation Flowshop Scheduling Problem. *International Journal of Production Research*, *51*(17), 5292–5308. https://doi.org/10.1080/00207543.2013.807955

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2015). Heuristics and metaheuristics for the

distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, *169*, 76–88. https://doi.org/10.1016/j.ijpe.2015.07.027

Huang, J.-P., Pan, Q.-K., & Gao, L. (2020). An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation*, *59*, 100742. https://doi.org/10.1016/j.swevo.2020.100742

Huang, J.-P., Pan, Q.-K., Miao, Z.-H., & Gao, L. (2021). Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times. *Engineering Applications of Artificial Intelligence*, *97*, 104016. https://doi.org/10.1016/j.engappai.2020.104016

Huang, Y.-Y., Pan, Q.-K., Huang, J.-P., Suganthan, P. N., & Gao, L. (2021). An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem. *Computers & Industrial Engineering*, *152*, 107021.

Jeong, I.-J., & Leon, V. J. (2002). A distributed scheduling methodology for a two-machine flowshop using cooperative interaction via multiple coupling agents. *Journal of Manufacturing Systems*, *21*(2), 126–139.

Jia, H. Z., Fuh, J. Y. H., Nee, A. Y. C., & Zhang, Y. F. (2002). Web-based Multi-functional Scheduling System for a Distributed Manufacturing Environment. *Concurrent Engineering*, *10*(1), 27–39. https://doi.org/10.1177/1063293X02010001054

Jia, H. Z., Nee, A. Y. C., Fuh, J. Y. H., & Zhang, Y. F. (2003). A modified genetic algorithm for distributed scheduling problems. *Journal of Intelligent Manufacturing*, *14*(3–4), 351–362. https://doi.org/10.1023/A:1024653810491

Jing, X.-L., Pan, Q.-K., Gao, L., & Wang, Y.-L. (2020). An effective Iterated Greedy algorithm for the distributed permutation flowshop scheduling with due windows. *Applied Soft Computing*, *96*, 106629.

Khare, A., & Agrawal, S. (2020). Effective heuristics and metaheuristics to minimise total tardiness for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 1–17.

Komaki, M., & Malakooti, B. (2017). General variable neighborhood search algorithm to minimize makespan of the distributed no-wait flow shop scheduling problem. *Production Engineering*, *11*(3), 315–329.

Lee, C.-Y., Cheng, T. C. E., & Lin, B. M. T. (1993). Minimizing the Makespan in the 3-Machine Assembly-Type Flowshop Scheduling Problem. *Management Science*, *39*(5), 616–625. https://doi.org/10.1287/mnsc.39.5.616

Lei, D., Su, B., & Li, M. (2020). Cooperated teaching-learning-based optimisation for distributed two-stage assembly flow shop scheduling. *International Journal of Production Research*, 1–14. https://doi.org/10.1080/00207543.2020.1836422

Lei, D., & Wang, T. (2020). Solving distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memeplex grouping. *Engineering Optimization*, *52*(9), 1461–1474. https://doi.org/10.1080/0305215X.2019.1674295

Li, H., Li, X., & Gao, L. (2020). A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem. *Applied Soft Computing*, 106946.

Li, W., Li, J., Gao, K., Han, Y., Niu, B., Liu, Z., & Sun, Q. (2019). Solving robotic distributed flowshop problem using an improved iterated greedy algorithm. *International Journal of Advanced Robotic Systems*, *16*(5), 1729881419879819.

Li, Y., Li, X., Gao, L., & Meng, L. (2020). An improved artificial bee colony algorithm for distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup times. *Computers & Industrial Engineering*, *147*, 106638. https://doi.org/10.1016/j.cie.2020.106638

Li, Y., Li, X., Gao, L., Zhang, B., Pan, Q.-K., Tasgetiren, M. F., & Meng, L. (2020). A discrete artificial bee colony algorithm for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 1–20. https://doi.org/10.1080/00207543.2020.1753897

Lin, S.-W., & Ying, K.-C. (2016). Minimizing makespan for solving the distributed no-wait flowshop scheduling problem. *Computers & Industrial Engineering*, *99*, 202–209.

Lin, W.-C. (2018). Minimizing the Makespan for a Two-Stage Three-Machine Assembly Flow Shop Problem with the Sum-of-Processing-Time Based Learning Effect. *Discrete Dynamics in Nature and Society*, *2018*, 1–15. https://doi.org/10.1155/2018/8170294

Mao, J., Pan, Q., Miao, Z., & Gao, L. (2021). An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Systems with Applications*, *169*, 114495. https://doi.org/10.1016/j.eswa.2020.114495

Meng, T., & Pan, Q.-K. (2021). A distributed heterogeneous permutation flowshop scheduling problem with lot-streaming and carryover sequence-dependent setup time. *Swarm and Evolutionary Computation*, *60*, 100804. https://doi.org/10.1016/j.swevo.2020.100804

Meng, T., Pan, Q.-K., & Wang, L. (2019). A distributed permutation flowshop scheduling problem with the customer order constraint. *Knowledge-Based Systems*, *184*, 104894.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, *37*(4), 754–768. https://doi.org/10.1016/j.cor.2009.06.019

Neufeld, J. S., Gupta, J. N. D., & Buscher, U. (2016). A comprehensive review of flowshop group scheduling literature. *Computers & Operations Research*, *70*, 56–74. https://doi.org/10.1016/j.cor.2015.12.006

Ong, Y. S., & Keane, A. J. (2004). Meta-Lamarckian learning in memetic algorithms. *IEEE transactions on evolutionary computation*, *8*(2), 99–110.

Ozsoydan, F. B., & Sagir, M. (2021). Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flowshop scheduling problem with sequence dependent setup times: a case study at a manufacturing plant. *Computers & Operations Research*, *125*, 105044. https://doi.org/10.1016/j.cor.2020.105044

Pan, J.-Q., Zou, W.-Q., & Duan, J.-H. (2018). A Discrete Artificial Bee Colony for Distributed Permutation Flowshop Scheduling Problem with Total Flow Time Minimization. In *2018 37th Chinese Control Conference (CCC)* (pp. 8379–8383). IEEE. https://doi.org/10.23919/ChiCC.2018.8482716

Pan, Q.-K., Gao, L., & Wang, L. (2021). An Effective Cooperative Co-Evolutionary Algorithm for Distributed Flowshop Group Scheduling Problems. *IEEE Transactions on Cybernetics*, 1–14. https://doi.org/10.1109/TCYB.2020.3041494

Pan, Q.-K., Gao, L., Wang, L., Liang, J., & Li, X.-Y. (2019). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications*, *124*, 309–324.

Parente, M., Figueira, G., Amorim, P., & Marques, A. (2020). Production scheduling in the context of Industry 4.0: review and trends. *International Journal of Production Research*, *58*(17), 5401–5431. https://doi.org/10.1080/00207543.2020.1718794

Potts, C. N., Sevast'janov, S. V., Strusevich, V. A., Van Wassenhove, L. N., & Zwaneveld, C. M. (1995). The Two-Stage Assembly Scheduling Problem: Complexity and Approximation. *Operations Research*, *43*(2), 346–355. https://doi.org/10.1287/opre.43.2.346

Riahi, V., Chiong, R., & Zhang, Y. (2020). A new iterated greedy algorithm for no-idle permutation flowshop scheduling with the total tardiness criterion. *Computers and Operations Research*, *117*. https://doi.org/10.1016/j.cor.2019.104839

Ribas, I., Leisten, R., & Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, *37*(8), 1439–1454. https://doi.org/10.1016/j.cor.2009.11.001

Ruiz, R., Pan, Q.-K., & Naderi, B. (2019). Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega*, *83*, 213–222. https://doi.org/10.1016/j.omega.2018.03.004

Sang, H.-Y., Pan, Q.-K., Li, J.-Q., Wang, P., Han, Y.-Y., Gao, K.-Z., & Duan, P. (2019). Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion. *Swarm and evolutionary computation*, *44*, 64–73.

Shao, W., Shao, Z., & Pi, D. (2020). Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. *Knowledge-Based Systems*, *194*, 105527. https://doi.org/10.1016/j.knosys.2020.105527

Simchi-Levi, D., Kaminsky, P., Simchi-Levi, E., & Shankar, R. (2008). *Designing and managing the supply chain: concepts, strategies and case studies*. Tata McGraw-Hill Education.

Wang, K., Huang, Y., & Qin, H. (2016). A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown. *Journal of the Operational Research Society*, *67*(1), 68–82. https://doi.org/10.1057/jors.2015.50

Wu, C.-C., Chen, J.-Y., Lin, W.-C., Lai, K., Bai, D., & Lai, S.-Y. (2019). A two-stage three-machine assembly scheduling flowshop problem with both two-agent and learning phenomenon. *Computers & Industrial Engineering*, *130*, 485–499. https://doi.org/10.1016/j.cie.2019.02.047

Wu, C.-C., Chen, J.-Y., Lin, W.-C., Lai, K., Liu, S.-C., & Yu, P.-W. (2018). A two-stage three-machine assembly flow shop scheduling with learning consideration to minimize the flowtime by six hybrids of particle swarm optimization. *Swarm and Evolutionary Computation*, *41*, 97–110. https://doi.org/10.1016/j.swevo.2018.01.012

Xiong, F., & Xing, K. (2014). Meta-heuristics for the distributed two-stage assembly scheduling problem with bi-criteria of makespan and mean completion time. *International Journal of Production Research*, *52*(9), 2743–2766. https://doi.org/10.1080/00207543.2014.884290

Xiong, F., Xing, K., Wang, F., Lei, H., & Han, L. (2014). Minimizing the total completion time in a distributed two stage assembly system with setup times. *Computers & Operations Research*, *47*, 92–105. https://doi.org/10.1016/j.cor.2014.02.005

Yang, S., & Xu, Z. (2020). The distributed assembly permutation flowshop scheduling problem with flexible assembly and batch delivery. *International Journal of Production Research*, 1–19. https://doi.org/10.1080/00207543.2020.1757174

Ying, K.-C., & Lin, S.-W. (2018). Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks. *Expert Systems with Applications*, *92*, 132–141. https://doi.org/10.1016/j.eswa.2017.09.032

Ying, K.-C., Lin, S.-W., Cheng, C.-Y., & He, C.-D. (2017). Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems. *Computers & Industrial Engineering*, *110*, 413–423.

Ying, K.-C., Pourhejazy, P., Cheng, C.-Y., & Syu, R.-S. (2020). Supply chain-oriented permutation flowshop scheduling considering flexible assembly and setup times. *International Journal of Production Research*, *58*(20), 1–24. https://doi.org/10.1080/00207543.2020.1842938

Zhang, G., & Xing, K. (2018). Memetic social spider optimization algorithm for scheduling

two-stage assembly flowshop in a distributed environment. *Computers & Industrial Engineering*, *125*, 423–433. https://doi.org/10.1016/j.cie.2018.09.007

Zhang, G., & Xing, K. (2019). Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion. *Computers & Operations Research*, *108*, 33–43.

Zhang, G., Xing, K., & Cao, F. (2018). Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion. *Engineering Applications of Artificial Intelligence*, *76*, 96–107.

Zhang, G., Xing, K., Zhang, G., & He, Z. (2020). Memetic Algorithm With Meta-Lamarckian Learning and Simplex Search for Distributed Flexible Assembly Permutation Flowshop Scheduling Problem. *IEEE Access*, *8*, 96115–96128. https://doi.org/10.1109/ACCESS.2020.2996305

Zhang, X., Li, X. T., & Yin, M. H. (2020). An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem. *International Journal of Bio-Inspired Computation*, *15*(2), 113. https://doi.org/10.1504/IJBIC.2020.106443

Zhao, F., Zhang, L., Cao, J., & Tang, J. (2021). A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Computers & Industrial Engineering*, *153*, 107082.

Zhao, F., Zhao, L., Wang, L., & Song, H. (2020). An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion. *Expert Systems with Applications*, *160*, 113678.